UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 204 *Numerical methods*

Newton's method

Douglas Wilhelm Harder, LEL, M.Math.
dwharder@uwaterloo.ca
dwharder@gmail.com

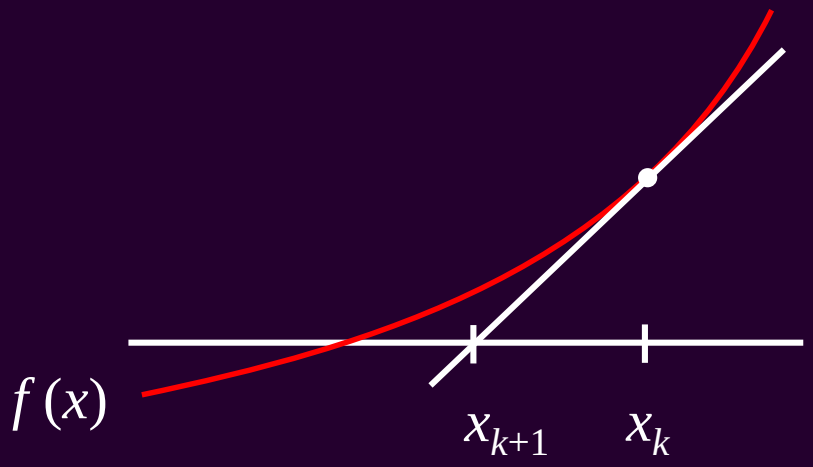
Introduction

- In this topic, we will
  - Describe and derive Newton's method
  - Deduce how quickly the algorithm converges
  - Consider when it may not converge
  - Look at an example
  - Describe automatic differentiation
  - Consider an implementation

Solutions to equations

- Suppose we have a real-valued function $f(x)$ of real variable and suppose that x_k is close to a root
  - Approximate the function at that point by a linear polynomial

$$f(x_k) + f^{(1)}(x_k)(x - x_k)$$



Solutions to equations

- What is the root of the linear polynomial?

$$f\left(x_k\right)+f^{(1)}\left(x_k\right)\left(x-x_k\right)=0$$

$$f^{(1)}\left(x_k\right)\left(x-x_k\right)=-f\left(x_k\right)$$

$$x-x_k=-\frac{f\left(x_k\right)}{f^{(1)}\left(x_k\right)}$$

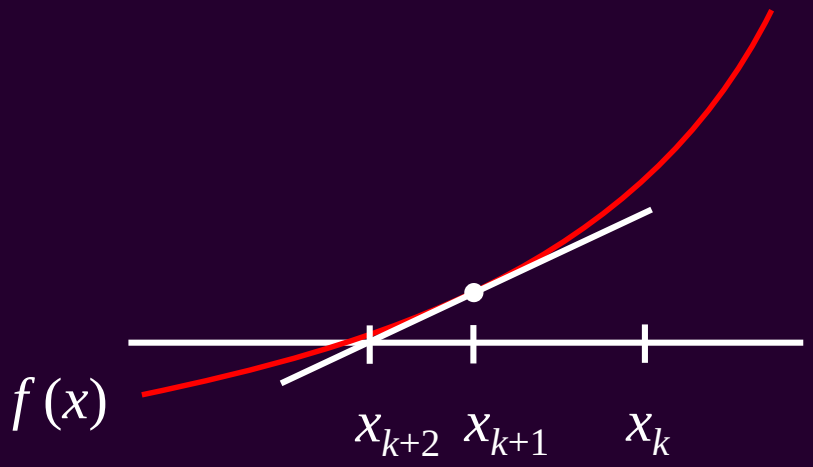
$$x=x_k-\frac{f\left(x_k\right)}{f^{(1)}\left(x_k\right)}$$

Solutions to equations

- Thus, if x_k is an approximation of the root, then

$$x_{k+1} \leftarrow x_k - \frac{f(x_k)}{f^{(1)}(x_k)}$$

should be a better approximation of the root



Solutions to equations

- Question: How much better?
  - If x_k is an approximation of a root r, the error is $r - x_k$

$$f(r) = f(x_k) + f^{(1)}(x_k)(r - x_k) + \frac{1}{2} f^{(2)}(\xi)(r - x_k)^2$$

$$0 = f(x_k) + f^{(1)}(x_k)(r - x_k) + \frac{1}{2} f^{(2)}(\xi)(r - x_k)^2$$

$$0 = \frac{f(x_k)}{f^{(1)}(x_k)} + r - x_k + \frac{1}{2}\frac{f^{(2)}(\xi)}{f^{(1)}(x_k)}(r - x_k)^2$$

$$0 = r - \left(x_k - \frac{f(x_k)}{f^{(1)}(x_k)} \right) + \frac{1}{2}\frac{f^{(2)}(\xi)}{f^{(1)}(x_k)}(r - x_k)^2$$

Solutions to equations

- Thus,

$$r - \left(x_k - \frac{f(x_k)}{f^{(1)}(x_k)} \right) = -\frac{1}{2} \frac{f^{(2)}(\xi)}{f^{(1)}(x_k)} (r - x_k)^2$$

$$r - x_{k+1} = -\frac{1}{2} \frac{f^{(2)}(\xi)}{f^{(1)}(x_k)} (r - x_k)^2$$

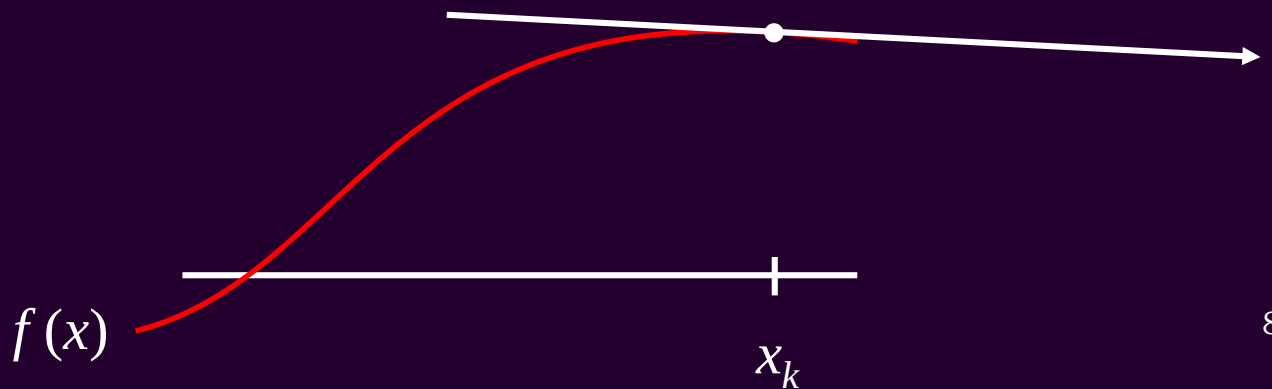
- Therefore, assuming the first- and second-derivatives are sufficiently smooth and the first derivative is not zero,

  the error is approximately a constant
  times the previous error squared

Description of the error

- We will describe Newton's method as O(h^2)
  - If h was the error at one step,
    the error at the next step is a constant times h^2

- Will Newton's method always converge?
  - If there is no root, it certainly will not converge: $x^2 + 1$
  - If the derivative at x_k is close to zero,
    the next approximation may be far away…even infinity



Example

- Find the first root of $2e^{-2x} - e^{-x}$ starting with $x_0 = 0.0$
  - The solution is ln(2)

k	x_k	$r - x_k$	$-\frac{1}{2}\frac{f^{(2)}(x_{k-1})}{f^{(1)}(x_{k-1})}(r - x_{k-1})^2$
0	0	0.6931	
1	0.333333333333333	0.3598	0.5605
2	0.565398476300347	0.1277	0.1641
3	0.672481930308809	0.02067	0.02273
4	0.692525169443129	0.0006220	0.0006320
5	0.693146600734152	0.0000005798	0.0000005801
6	0.693147180559441	0.000000000005045	0.000000000005043
7	0.693147180559945	0	

Implementation

- Newton's method requires us to supply both the function and the derivative
  - Automatic differentiation is the algorithm of taking a function and creating a function that gives the derivative exactly

```
double f( double x ) {
    return 2*exp(-2*x) - exp(-x);
}

double df( double x ) {
    return -4*exp(-2*x) + exp(-x);
}
```

Implementation

```
double newton( double  f( double x ), double df( double x ),
               double x0, double eps_step, double eps_abs,
               unsigned int max_iterations ) {
    for ( unsigned int k{0}; k < max_iterations; ++k ) {
        double x1{ x0 - f(x0)/df(x0) };   % Newton's method

        if ( !std::isfinite( x1 ) ) {
            return NAN;
        }

        if ( (std::abs( x1 - x0 ) < eps_step)
            && (std::abs( f(x1) ) < eps_abs) ) {
            return x1;
        }

        x0 = x1;
    }

    return NAN;
}
```

$$x_{k+1} \leftarrow x_k - \frac{f\left(x_k\right)}{f^{(1)}\left(x_k\right)}$$

$$\left|x_{k+1} - x_k\right| < \varepsilon_{\text{step}} \text{ and } \left|f\left(x_{k+1}\right)\right| < \varepsilon_{\text{abs}}$$

C/C++ Code is provided to demonstrate the straight-forward nature of these algorithms and not required for the examination

Summary

- Following this topic, you now
  - Have reviewed Newton's method
  - Understand that the error is $O(h^2)$ where we derived this using the first-order Taylor series
  - Know that the algorithm may not converge
  - Have seen an example of the algorithm in practice
  - Have learned about automatic differentiation and considered an implementation of this algorithm

References

[1] https://en.wikipedia.org/wiki/Newton%27s_method

Acknowledgments

Staisha Neville for noting that in estimating the error of Newton's method on Slide 9, I indicated I was using $f^{(2)}(r)/f^{(1)}(r)$ when in fact I was using the ratio $f^{(2)}(x_{k-1})/f^{(1)}(x_{k-1})$

Colophon

These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`. Mathematical equations are prepared in MathType by Design Science, Inc.

Examples may be formulated and checked using Maple by Maplesoft, Inc.

The photographs of flowers and a monarch butter appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens in October of 2017 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer

These slides are provided for the ECE 204 *Numerical methods* course taught at the University of Waterloo. The material in it reflects the author's best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.